Preserving HTTP Sessions in Vehicular Environments

Yibei Ling, *Senior Member, IEEE,* Wai Chen, *Senior Member, IEEE,* T. Russell Hsing, *Fellow, IEEE,* and Onur Altintas, *Member, ACM, IEEE*

***Abstract*— Wireless Internet in the in-vehicle environment is an evolving reality that reflects the gradual maturity of wireless technologies. Its complexity is reflected in the diversity of wireless technologies and dynamically changing network environments. The ability to adapt to the dynamics of such environments and to survive transient failures due to network handoffs are fundamentally important in failure-prone vehicular environments. In this paper we identify several new issues arising from network heterogeneity in vehicular environments and concentrate on designing and implementing a network-aware prototype system that supports HTTP session continuity in the presence of network volatility, with the emphasis on the following specifically tailored features: (1) automatic and transparent HTTP failure recovery, (2) network awareness and adaptation, (3) application-layer preemptive network handoff. Experimental results gathered from real application environments based on CDMA *1xRTT* and IEEE 802 networks are presented and analyzed.**

***Index Terms*— Network hysteresis, preemptive network handoff, heterogeneous network, session-level HTTP failure recovery, and packet-level HTTP failure recovery.**

I. Introduction

RAPID technology advancement in mobile communication has significantly changed our way of communication and information exchange. The proliferation of various portable devices such as laptop computers, personal digital assistants (PDA), cellular phones, and the ubiquity of various cellular networks such as CDMA, CDPD, GSM, and wireless LAN (WLAN) IEEE 802.11a/b/g, offer a powerful mobile computing platform that completely frees devices/computers from being tethered to the wired line. Thus convergence and integration of Internet [7] and wireless technologies spark new ways of using Internet and spurs a growing demand for a wide range of telematics applications.

Ensuring HTTP session continuity in vehicular environments is of paramount importance to vehicular Internet access, since network connectivity is inherently unreliable due to the presence of network handoff and the existence of blind coverage spots. However, the issue of HTTP session continuity has been largely overlooked: it appears to be irrelevant to real applications since driving and web surfing at the same time are generally considered as a risk distraction. Such a viewpoint is reflected in the comment "don't walk while telnet'ing" by Henning Schulzrinne [25].

The emerging telematics applications, especially a wide range of rear-seat applications including HTTP-based application [7] highlight the importance of HTTP session continuity. However, the gap between the promise of in-vehicle Internet access and reality remains wide, mainly reflecting in the following distinct aspects:

1) the presence of multiple networks with different characteristics in overlapping coverage areas;
2) the inherently unreliable network connectivity in vehicular environments due to the presence of blind coverage spots, network handoffs and high vehicle mobility;
3) existing applications are essentially network-oblivious and lack the ability to cope with dynamically changing network environments.

In a vehicular environment, a mobile host (MH hereafter) constantly changes its geographic location, resulting in the switching of coverage responsibility from one base station to another. Such network switch, called network handoff, causes the MH to be temporarily disconnected to the network, thus resulting in a disruption of on-going sessions. The adverse effect of handoff upon system performance could become prominent in highly mobile vehicular environments [11]. Thus, network handoff is a major mobility issue that has been receiving a great deal of research interest in both industry and academia.

From the perspective of a MH, the network handoff can be classified into two distinct categories: (1) horizontal handoff and (2) vertical handoff [26]. A horizontal handoff refers to a network switch taking place in homogeneous network environments. For instance, a handoff occurs during subnet-crossing in a WLAN network. A vertical handoff refers to a network switch taking place in heterogeneous network environments, such as a handoff between a CDMA cellular network and a Wireless LAN (WLAN) network.

The vehicular Internet access is more prone to transient network failures than wired line Internet access mainly for two reasons. First, handoffs induced by vehicle mobility are far more common than pedestrian mobility due to high vehicle mobility. Second, coverage of wireless network is far from ubiquitous: the presence of blind coverage spots becomes a source of losing network connection in vehicular environments. Another equally important aspect is the efficiency of HTTP failure recovery, especially for long HTTP session (downloading large files) in relatively low bandwidth wireless

Manuscript received: September 20, 2003; revised: November 06, 2005; accepted: October 13, 2006

Yibei Ling, Wai Chen, and T. Russell Hsing are with Applied Research, Telcordia Technologies, One Telcordia Drive, Piscataway, NJ 08854, USA

Emails: {lingy,wchen,trh}@research.telcordia.com

Onur Altintas is with Research & Developement Division, Toyota InfoTechnology Center, Co., Ltd., 6-6-20 Akasaka, Minato-ku, Tokyo, 107-0052 Japan. This work was done when the author was at Toyota InfoTechnology Center, USA and also a visiting researcher at Telcordia Technologies

Email: onur@jp.toyota-itc.com

network environments. In a vehicular environment, when an HTTP session is punctuated by transient network failures, the HTTP session (a kind of TCP session) needs to be restarted from scratch. This could incur expensive network resource in wireless vehicular environments since transient network failures could be frequently encountered events. Therefore, an efficient and transparent HTTP failure recovery is an essential feature required in failure-prone vehicular environments.

Wireless vehicular environments are inherently heterogeneous, consisting of different types of wireless technologies with different characteristics. It is well known that the CDMA and WLAN networks differ significantly in terms of bandwidth capacity and coverage range: the CDMA network is of relative low bandwidth capacity but has a wide coverage area, while the WLAN network is of high bandwidth capacity but has a narrow coverage area [6], [4], [16], [18].

The coexistence of CDMA and WLAN networks in overlapping coverage areas gives rise to a fundamental phenomenon in heterogeneous network environments called *network hysteresis* (access history dependency), which refers to the inherent tendency of an established TCP (HTTP) session to stay with a network once the session has been established over the network.

This situation can be exemplified as follows: a MH with multiple network interfaces initiates a (TCP or HTTP) session over a low-bandwidth network such as the CDMA network, the MH then moves into the overlapping coverage area with a high-bandwidth network such as WLAN network. Despite the presence of the high-bandwidth network, the MH continues to stay with the low-bandwidth network (CDMA). It is because that (1) the CDMA network has a wider coverage area as compared to the WLAN network, (2) there is an inherent tendency to maintain connectivity over a network after the initial (TCP or HTTP) session has been established (inherent network environment obliviousness).

The phenomenon of the *network hysteresis*, which arises only in heterogeneous wireless network environments, carries an adverse performance implication for the in-vehicular applications: it prevents the MH from utilizing the best available network resource, thereby impairing system performance in inherently heterogeneous vehicular network environments. To our best knowledge, the impact of network hysteresis on system performance has not been considered before.

In this paper we introduce the preemptive handoff to address the problems arising from the network hysteresis. The main idea behind the preemptive handoff is to dynamically reselect the best available network interface in the presence of multiple network interfaces. This involves automatically relinquishing a lower-bandwidth connection and reestablishing a higher-bandwidth connection while at the same time maintaining session continuity. Preemptive handoff in spirit is similar to the downward vertical handoff which is defined as network handoff from a low-bandwidth network to a high-bandwidth network [26]. However, the proposed preemptive handoff differs markedly from vertical or horizontal handoff in its goal to optimize system throughput, in addition to ensuring session continuity during handoffs. In this paper we present an architectural framework as well as a prototype system based on this framework, stressing particularly on the following aspects.

- supporting HTTP session continuity in the presence of handoffs;
- network awareness and network adaptation;
- carrier-independence;
- automatic and efficient HTTP failure recovery;
- access transparence.

The remainder of this paper is organized as follows: Section 2 gives a brief review of related mobility approaches in the literature, with the focus on mobile IP and Session Initiation Protocol (SIP hereafter). Section 3 presents a system architecture, as well as its implementation, for HTTP failure recovery. Section 4 describes our heterogeneous network access testbed with cellular CDPD and CDMA networks, and WLAN networks. The experimental results are presented and discussed. Section 5 concludes the paper with possible future extensions.

II. Overview of Related Work

In this section we first introduce the notions of HTTP session, and session-level, and packet-level HTTP failure recovery, followed by a review of Mobile IP (MIP hereafter) and Session Initiation Protocol (SIP hereafter) . We then discuss the inadequacy of SIP and MIP in dealing with HTTP session in a practical setting. We also briefly discuss the Wireless Application Protocol (WAP) and touch on the Tarantella software system. Finally, we look beyond MIP and SIP for solution to problems stemming from network heterogeneity.

Definition 1: An HTTP session consists of an HTTP request and the HTTP response to that request. ▲

An HTTP request includes both explicit and implicit HTTP request. An explicit HTTP request is initiated manually, whereas an implicit HTTP request, as an ancillary event triggered by an explicit HTTP request, is initiated transparently by the Web client. To put more intuitively, an HTTP session is started with a request initiated from a Web client to a Web server and terminated with the reception of the entire response from the origin Web server. The following definitions are given to distinguish the two granular levels of HTTP failure recovery.

Definition 2: A session-level HTTP failure recovery is atomic or indivisible with respect to a given point of attachment. ▲

Session-level HTTP failure recovery carries an all-or-nothing implication, meaning that an HTTP failure cannot be partially recovered. For instance, downloading an HTTP file either completes or fails with respect to a given point of attachment (network). Session-level HTTP failure recovery could be relatively palatable in a wired network environment because of relatively high bandwidth. However, it could become extremely annoying when network handoffs occur in the midst of long-lived HTTP sessions, especially when most of the transfer has taken place. Therefore, the weakness of session-level HTTP failure recovery is its inefficiency

of network utilization. A failure recovery requires that file transferring be restarted from scratch, which could be very costly in relatively low bandwidth network environments. In addition, a frequent handoff in failure-prone vehicular environments could result in a frequent failure recovery, which is likely to form an endless cycle where failures and premature recoveries are interlocked. As a result, session-level HTTP failure recovery by nature cannot guarantee the progressiveness in the presence of frequent handoff. Packet-level HTTP failure recovery is thus proposed to address the deficiency of its session-level counterpart. Its definition is given as below:

Definition 3: A packet-level HTTP failure recovery is divisible with respect to the point of attachment. ▲

Packet-level HTTP failure recovery differs from the session-level counterpart in its ability to avoid HTTP session being restarted from scratch after a failure occurs. This means that constituent data packets obtained from different points of attachments (networks) can be seamlessly pieced together. The packet-level HTTP failure recovery improves upon the session-level counterpart in its failure-recovery efficiency, thus making it more suitable in failure-prone vehicular environments. Now we are in a position to review existing approaches in the literature, and evaluate their pros and cons in a practical setting.

Mobile IP (MIP) is a standard network-layer mobility protocol, allowing a MH to maintain session continuity when roaming to a different network [19], [11], [9], [13]. It is implemented via IP-in-IP encapsulation, IP tunneling, and IP decapsulation. Under the MIP scheme, the MH consists of (1) a fixed IP (primary IP or home address); (2) a care-of address that is changed with the change of point of attachment. When the MH moves to a new location (foreign network), it registers the IP address of a foreign network with the home agent located on the user's home network. The home agent is thereby able to transparently tunnel all packets to the user's current location via the IP-in-IP encapsulation. Upon receipt of data packets from the home agent, the foreign agent decapsulates packets and delivers them to the MH.

The most attractive feature of MIP is its application transparency. It supports mobility and session continuity without the need to modify existing applications. However, the benefits of the MIP come at a performance price: MIP requires performing run-time IP-in-IP encapsulation and decapsulation for each IP packet, and introduces triangle routing as well as an additional delay between home agent and foreign agent. As a result, mobile IP could increase the access latency by 45% in a typical campus environment [27]. Much research effort has focused on routing optimization in order to minimize the effect of costly triangle routing [13], [21], [3], [5], [12], [22], [23], and reduce the handoff time [26], [18].

Session Initiation Protocol (SIP) is widely used as the replacement of H.323 protocol for multimedia streaming, various UDP-based applications such as Internet conferencing, telephony, instant messaging and real-time event notification. Major components in SIP are SIP user agent and SIP proxy/redirect servers. Under the SIP scheme, a SIP user agent residing in the MH is responsible for updating the home SIP proxy server with the current location (IP address), and managing the SIP-aware applications. Each SIP user is addressed by a unique SIP identity (an email-like address), which is initially registered in a SIP registrar managed by the SIP proxy server, together with the location (IP address) of the MH. To communicate with a peer MH, the MH sends an INVITE message with the SIP identifier of the peer MH to the SIP proxy server, which in turn returns the current location (IP address) of the peer MH via dynamic binding (identifying the IP address through its SIP identifier). The MH can then use this IP address to directly send an INVITE message to the peer MH. Since it is being implemented on top of UDP or TCP, SIP is widely used as a signaling protocol for session establishment and management in mobile environments for terminal mobility and service mobility. It can also be used to support streaming-based session (VoIP) continuity such as RTP. However, it does not work very well for TCP-based applications [25], [27].

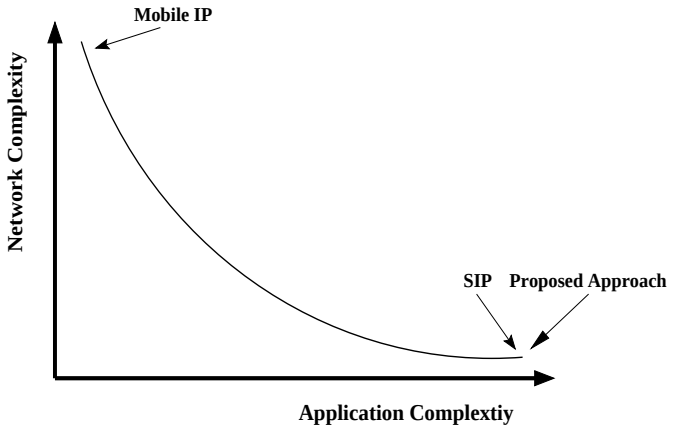

Fig. 1. Network Complexity vs. Application Complexity of Mobile IP and SIP

MIP and SIP are two distinct and competing paradigms for supporting mobility and session continuity, representing two opposite extremes in terms of network and application complexities. MIP has a high level of network complexity with a relatively low application complexity. In contrast, SIP has a relatively high level of application complexity with a minimum level of network complexity, as shown in Fig(1).

Market environment issues, however, are often perceived as the single most important challenge to mobile IP and SIP deployment because of the autonomy of service providers and wireless operators, which adds another level of complexity in vehicular Internet access. Deployment of Mobile IP and SIP proxy/redirect server is required to have a complete control over underlying wireless networks. This is only possible for wireless operators such as AT&T and Verizon [26]. Thus it could be inoperative under many conditions for wireless service subscribers/customers. For instance, a SIP user agent may be unable to receive a session initiation call from the SIP server managed by different service providers and wireless operators. The vertical handoff scheme proposed in [26] requires that

a home agent be able to multicast data packets to a group of of base stations in which a foreign agent resides, which is constrained by a variety of factors in practice. For those networks that we cannot control, it is impossible to put a foreign agent into their base stations. In reality, a vertical handoff normally could take more than 10 seconds (it will be shown later on), which could cause an unrecoverable TCP failure (MIP) because it exceeds TCP timeout [11], [20].

Tarantella is a commercial thin-client software system that enables enterprises to provide users anywhere with managed secure web-based access to critical corporate applications and services. The most attractive feature of the Tarantella system is its resumable applications. It allows users to log out of their webtops while keeping their applications running on servers. As a result, users can initiate a long-duration calculation, log out of the Tarantella system, and then find their results after reaching their destinations [2]. Our approach, on the other hand, addresses the problem associated with file downloading in heterogeneous wireless environments. It focuses on resumable file downloading and network environment awareness, rather than on resumable lengthy computation.

Wireless Application Protocol (WAP) aims to provide Internet content to mobile devices, pagers and other wireless terminals over low-bandwidth wireless networks [1]. The WAP stack is divided into five hierarchy levels: (1) wireless application environment (WAE); (2) wireless session protocol (WSP); (3) wireless transaction protocol (WTP); (4) Wireless Transport Layer Security (WTLS) and (5) wireless datagram protocol (WDP). WAP places its premium on the efficiency of wireless data transfer. But this efficiency comes at a price. In order to leverage the popularity of Web servers, WAP needs to first translate the HTML documents to the corresponding WML (wireless markup language) documents. Then it needs to convert the WML content into the binary format to further reduce the transmission size [1]. Such protocol transformation requires an additional processing step that considerably complicates the end-to-end information flow. The need for wireless efficiency of WAP is in fact gradually obviated by advances in wireless technologies such as 3G and IEEE 802.11b.

Our prototype system addresses different aspects of wireless data transfer than the Tarantella system and the WAP protocol. The difficulty of vehicular Internet access is reflected in the complexity of network heterogeneity, and the autonomy of service providers and wireless carriers. In addition, transient network failures due to the presence of blind coverage spots and vehicle speed should be one of the primary concerns in vehicular network environments.

To investigate the effect of vehicle speed on network connectivity and system throughput, we conducted a real-life experimental study. We used an ICMP-based probe with packet size of 64 bytes to poll *www.yahoo.com* continuously using the Verizon CDPD and CDMA *1xRTT* networks respectively. We then measured the round-trip time (RTT) of each probe. Figs(2)-(4) present the measurement results obtained from a 45-minute drive time during rush hour (9am-11am). Figs(3)-(5) represent the results obtained from a fixed-location test (a window office around 11:3am). Notice that in Figs(2)-(5), transient network disconnects were represented by zero round trip time for a better visualization.

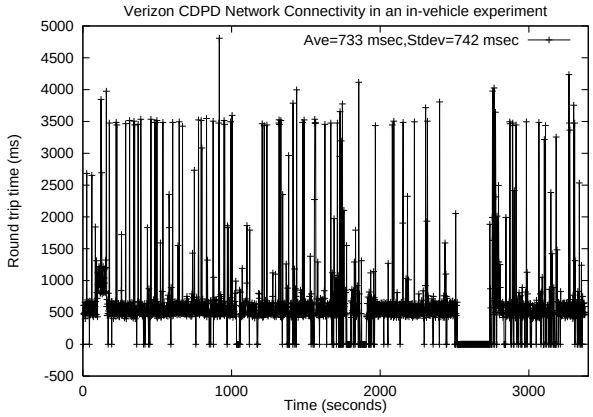

Fig. 2. In-vehicle Verizon CDPD connectivity

This comparison study showed that among the total 1689 samples taken, 203 packets were dropped in the in-vehicle CDPD connectivity experiment. This accounts for a 12% packet drop compared to the $194/5567 = 3\%$ packet drop rate in the fixed location. Also among the total 2401 samples taken, 58 packets were dropped in the in-vehicle CDMA connectivity experiment, accounting for a 2.4% packet drop compared to the $9/2500 = 0.3\%$ packet drop rate in the fixed location. We also observed that among the total 1689 probes in the in-vehicle CDPD connectivity experiment, $97(5.74\%)$ probes were responded with no network connectivity due to the existence of blind coverage spots during driving. By removing dropped packets and packets responded with no network connection, we used the mean and standard deviation to quantify the delay variability in Table 1.

TABLE 1
CONNECTIVITY IN FIXED-LOCATION AND IN-VEHICLE ENVIRONMENTS

	CDPD		*CDMA*	
	mean	stdev	mean	stdev
in-vehicle	$733ms$	$742ms$	$503ms$	$105ms$
fix-location	$677ms$	$566ms$	$508ms$	$81ms$

The real-life experimental results in Table 1 suggested that vehicle speed indeed introduces an additional delay variability (increased standard deviation of RTT in both CDPD and CDMA tests), and thus could be considered as a non-negligible cause of connectivity unreliability and performance deterioration. This research work is motivated by the fact that vehicular Internet access is an important part of telematics applications, especially for various rear-seat applications. We identify several important but overlooked factors that negatively affect vehicular Internet access, such as network hysteresis and frequently encountered network failures, and present a framework in an effort to mitigate these problems.

III. ARCHITECTURE DESCRIPTION

In this section, we focus on the design and implementation of a prototype system to support an efficient HTTP failure

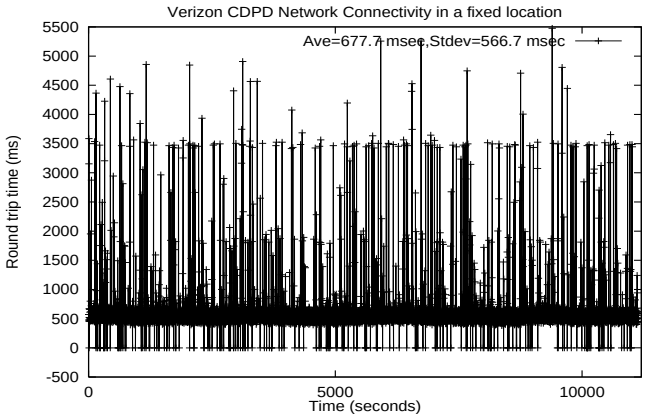

Fig. 3. Fixed-location Verizon CDPD connectivity

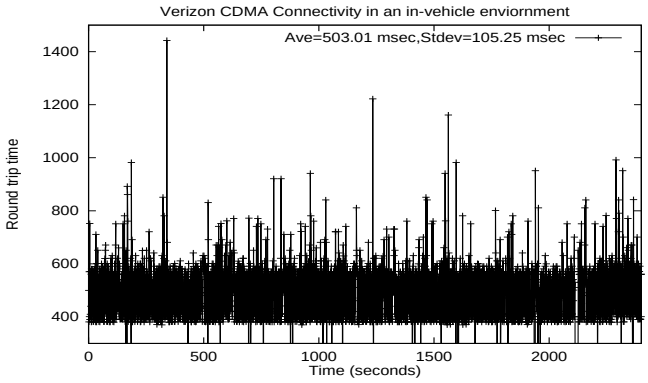

Fig. 4. In-vehicle Verizon CDMA connectivity

recovery, network awareness and network adaptation. It is worth noting that our prototype system is built on top of a Microsoft Window 2000 server and XP. As a service subscriber to the Verizon CDMA data service, we do not have control over the Verizon CDMA network as well.

It is well-known that Web servers and Web browsers are the fundamental architectural building blocks in the World Wide Web. Web servers are intrinsically stateless and each request is processed without any knowledge of previous requests. Any network failure will disrupt ongoing HTTP sessions, thus requiring the user to manually reestablish a connection to the same server. This HTTP failure recovery mechanism works well in the wired network but it does not sit well with the wireless vehicular environments because of the lack of two essential features:

- automatic and efficient HTTP failure recovery without human intervention,
- network awareness with ability to adapt to changing network conditions.

An automatic HTTP failure recovery means that the recovery is invoked as an ancillary event in response to a network failure, without any manual intervention on the part of the user. This feature, however, is not supported by Web browsers. Furthermore, the real challenge is to adequately address the problems related to vehicular environments while at the same

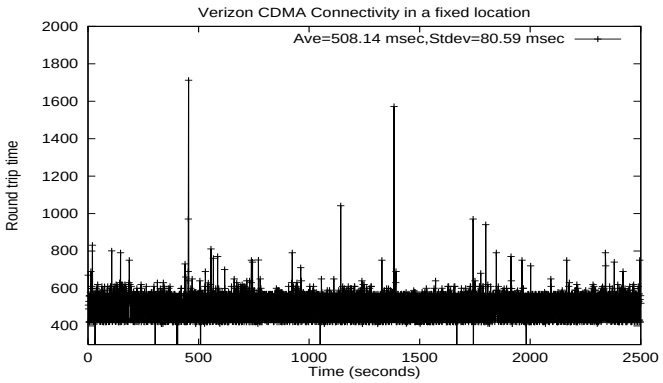

Fig. 5. Fixed-location Verizon CDMA connectivity

time keeping the existing Web server and Web client intact.

To meet these requirements, we propose a multi-tiered architecture as shown in Fig(6). This architecture consists of a client-side proxy and an information gateway (IGW hereafter), sitting between the Web browser and the Web server. The addition of the client-side proxy and the IGW is to provide a shield that supports automatic and transparent HTTP failure recovery while keeping existing Web server and Web browser intact. The entire prototype system is structured into the hierarchy functional layers. The client-side proxy subsystem is implemented through the layering of five types of technologies, and the IGW subsystem is implemented through the layering of four types of technologies. As a means of walling off complexity, the layer classification focuses on structural extensibility. As a result, a new layer could be added flexibly without needing to rewrite a significant chunk of infrastructure to support the requirement change.

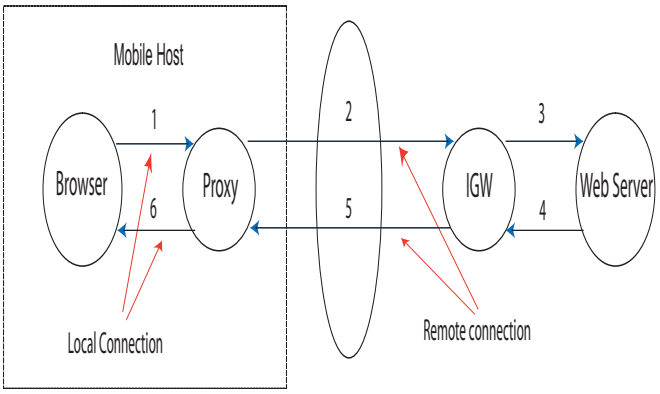

Fig. 6. Processing Flow

A detailed breakdown of the layering structure is shown in Fig(7). The top layer at MH is HTTP proxy that provides a transparent access interface to the Web browser by concealing complicated HTTP failure recovery and network awareness functionality. Its primary task is to intercept HTTP requests from the Web client and to split connectivity into HTTP local and remote connections. The HTTP session layer is to keep track of each ongoing HTTP session (byte-count and time stamp of each ongoing session) and to trap various system events generated by the network sensing layer. Upon

the receipt of event notification such as network failure, the HTTP session layer can automatically restore the affected HTTP sessions by exchanging session information with its counterpart at the IGW. The network sensing layer can capture changing network conditions and inform the HTTP session layer of such changes in a timely fashion. In addition, it is able to dynamically select the best network interface in the presence of multiple networks, providing network awareness and adaptation.

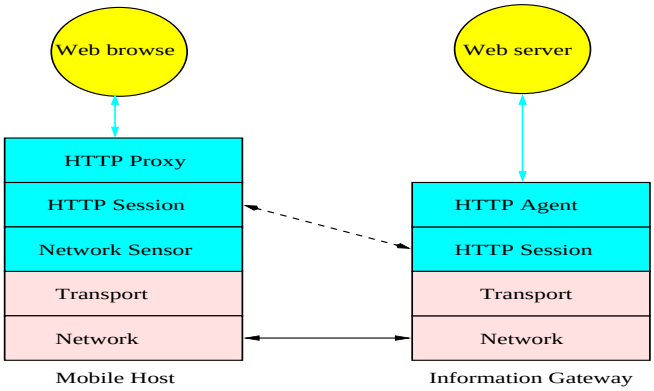

Fig. 7. Logical Layering Hierarchy

The HTTP session layer at the IGW works in tandem with its counterpart at the MH, providing the stateful management of ongoing HTTP session. In the event of network failures, the HTTP session layer at the MH automatically resynchronizes with the IGW with the stateful information about affected ongoing session without human intervention. This allows the HTTP session layer at IGW to pick up the data stream at point of interruption. Finally, the HTTP agent layer at the IGW is to directly interface with Web servers.

It is worth noting that FTP protocol with restart option and HTTP restart in HTTP 1.1 specification could be used to support packet-level failure recovery. Rather than focusing exclusively on failure recovery protocols, our approach is based on a multi-tier system architecture, focusing not only on automatic packet-level HTTP failure recovery, but also on network awareness and adaptation. Our prototype system is widely applicable, irrespective of the protocols being used. In the following sections, we will go through each layers of the hierarchy structure of each subsystem in detail.

A. Network Sensing Layer & HTTP Session Layer

The network sensing layer at the MH provides the network-aware capability, thus serving as a means to inform the HTTP session layer of changes in network conditions (see Fig(8) for details).

The ability to rapidly discern changes in network conditions and to pinpoint the root cause of such changes is an essential network-aware feature. It also plays a crucial role in automatic failure recovery. To this end, we consider two application-layer mechanisms as the core of network sensing layer: (1) event-driven scheme and (2) polling-based scheme. These two complementary schemes are used in parallel in the prototype

Listing 1. Event Capturing

```
try {
  SocketConnection
  while ( EOF == false)
  {
     WriteSocket
     nRead=ReadSocket
     bytecount = bytecount+nRead;
     if a higher bandwidth network found
         return PREEMPTIVE_EVENT;
   }
   return;
  }
  catch(SocketException)
  {
    return GetExceptionCode;
  }
```

system, serving distinct roles in detecting various types of network failures.

The event-driven scheme is used to capture various network events taking place during the course of HTTP sessions. From an application-layer perspective, ongoing HTTP sessions are fundamentally *socket*-related. As a result, socket-related exceptions are raised when network failures (events) occur during HTTP sessions. The idea underlying the event-driven scheme is to take the advantage of the *socket*-related exception handling to trap various network failures (events) during the course of HTTP sessions and to identify the root cause of such (events) failures.

The polling-based scheme is used to gather the overall network conditions via periodic polling of MH's network interfaces. This scheme is particularly useful for detecting changes in network environment when the MH moves into the coverage area of a new WLAN network in the absence of active HTTP sessions. In our implementation, an asynchronous thread-based *poller* is used to periodically retrieve network interface information on the MH, using the *WSAIoctl* function with SIO_GET_INTERFACE_LIST option in Microsoft Platform SDK. The presence or disappearance of a wireless network is detected through periodic polling by comparing the current status with the previous one. A notification to the HTTP session layer will be generated to report such a change.

The pseudocode in Listing 1 illustrates our implementation of the event-driven scheme to capture random network events during an HTTP session. The code within the *try block* section corresponds to an ongoing HTTP session being executed. A network event taking place during the course of a HTTP session will trigger an exception in the *try block* section. The code within the *catch block* (exception handler) is thus invoked. The root cause of the network event is then identified by examining the corresponding exception error code. In addition, we define a message code *PREEMPTIVE_EVENT* to represent the presence of a high bandwidth network during an HTTP session, which is detected by a thread-based *poller* via periodic polling. The presence of such events results in a preemptive handoff (switching to a relatively higher bandwidth network) to better utilize the available bandwidth resource. The

Listing 2. Exception Handling

```
switch (nRet)
{
  case 0:
    request = GetRequest();
    break;
  case PREEMPTIVE_EVENT:
    Preemptive();
    break;
  case WSAEHOSTDOWN:
  case WSAECONNABORTED:
  case WSAECONNRESET:
  case WSAENETDOWN:
  case WSAENETUNREACH:
  case WSAENETRESET:
  case WSATRY_AGAIN:
  case WSANO_RECOVERY:
  case WSAEADDRNOTAVAIL:
    Handoff();
    break;
  default;
    break;
}
```

pseudocode in Listing 2 describes how to identify the cause of network events (failures) and how to invoke a proper function based on the nature of network event. A variety of error codes representing various transient network failures during an HTTP sessions is identified and provided in the above code snippet.

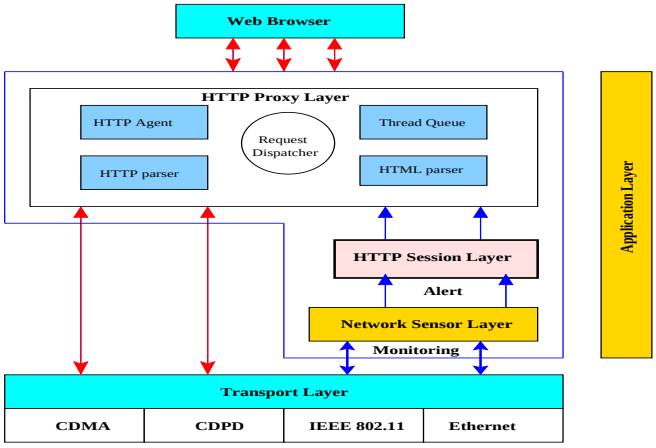

Fig. 8. System architecture of mobile host

Detection delay is an important metric for measuring the responsiveness of a MH to change in the network environment. An increased polling frequency generally results in a decreased detection delay, as shown in Fig(9). On the other hand, an excessive polling takes up run-time kernel resources [15]. Thus it is important to select an appropriate polling frequency to balance system responsiveness and run-time resources. The following theorem establishes a connection between detection delay, polling frequency and change rate of network environment, by assuming that change in the network condition follows the Poisson process with intensity λ.

Theorem 1: Let $T > 0$ be a polling interval, and change in network conditions be a Poisson process with intensity λ, then the long-run mean average detection delay, $E(D)$, is bounded by

$$E(D) < \frac{(T\lambda)^2 - 2T\lambda + 2 - 2\exp(-T\lambda)}{2T(\lambda)^2}, \qquad (1)$$

where $E(.)$ is the expectation function. ▲

Proof of Theorem is given in Appendix. Notice that polling interval is a tunable system parameter. Fig(9) presents the dependency of detection delay on polling frequency and network environment change rate. Assuming that $T = 1/\lambda = 10$ seconds, that is, the polling interval is the same as the mean interarrival time of network condition of 10 seconds, then the long-run mean average detection delay based on Theorem 1 is less than $10 * (0.5 - \exp(-1)) = 1.34$ seconds. In our implementation, we set the polling time as 10 seconds, by taking into account the responsiveness and the resource usage.

The primary function of HTTP session layer is to keep track of HTTP sessions, including the byte-count of each ongoing sessions as illustrated in the pseudocode in Listing 1. The byte-count information plays a crucial role for packet-level HTTP failure recovery.

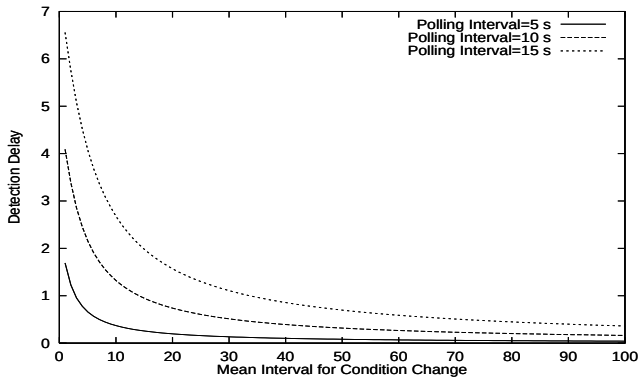

Fig. 9. Detection Delay vs. Network Change Rate

B. HTTP Proxy Layer

The function of the HTTP proxy layer is to provide Web access transparency as well as to add a new functional layer that supports network adaptation and packet-level failure recovery, without the need to modify existing Web server and Web browser.

The proxy layer splits each HTTP request initiated from the Web browser into two separate HTTP connections: the local one to the Web client and the remote one to the IGW, as shown in Fig(6). Upon receipt of an HTTP request from the Web browser, the client-side proxy transparently redirects the request to the IGW, which in turn forwards the request to an origin Web server. The local connection between the proxy and the Web browser is impervious to network failure, whereas the remote connection between the client-side proxy and the IGW is vulnerable to network failure. Such a splitting connectivity provides a clear separation of local and remote HTTP sessions,

allowing us to gracefully handle network failures while at the same time maintaining user-perceived HTTP continuity. Additionally, the packet-level HTTP failure recovery is implemented into the HTTP proxy layer in which fragmented data is properly pieced together to avoid restarting the HTTP session from scratch in the presence of network failures. Thus it ensures the efficiency of HTTP failure recovery without the need to modify existing Web client and server.

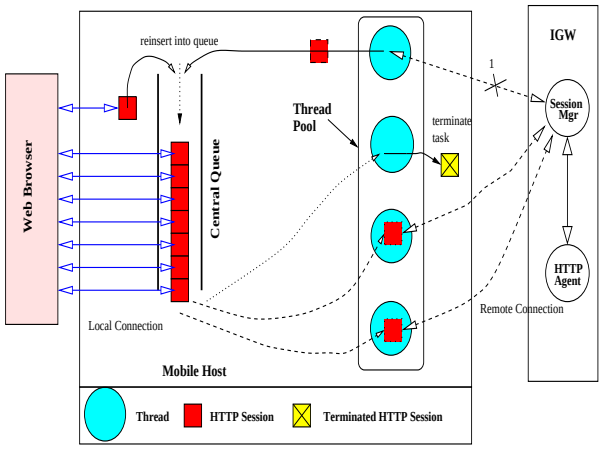

Fig. 10. Thread Pool and Task Queue

An instance of the HTTP session class, which is derived directly from Microsoft foundation class *CObject*, is created for per HTTP request and is terminated (released) when the HTTP session has completed. The class object contains the two socket objects for local and remote connections. Since it is connected to the Web browser via loopback IP address 127.0.0.1, the local connection is persistent across the life time of the object. The remote connection, however, is independent of the life time of the thread object; it could be terminated and recreated several times across the duration of the class object, depending on network conditions.

We implemented a thread pool architecture [17], [14] to efficiently manage worker threads. The idea behind the thread pool is to reuse worker threads. Each worker thread in the pool thread can be recycled, thereby improving the system performance by avoiding repeated and costly thread creation. Upon startup, a fixed number of worker threads are created waiting for tasks. A thread object in the queue is extracted by an idle worker thread in the pool in the FIFO fashion. It can be seen from Fig(10) that a newly created thread object, which corresponds to an HTTP request initiated by the Web client, is immediately put at the end of a queue waiting for processing. As the worker threads finish with old tasks and become available, a thread object is extracted from the queue. When a network failure (handoff) occurs during HTTP sessions, affected thread objects are placed at the end of the queue again waiting for the network to recover and will be reprocessed by an idle worker thread in the thread pool. The lines 1 and 2 in Fig(10) represent the reprocessing flow. The size of the thread pool (number of worker threads) in our implementation is set to four for a balanced threading concurrency and context switching overhead [17], [14].

Listing 3. GetExServerConnection

```
GetExServerConnection()
{
   parse HTTP request
   extract session offset
   connect to a Web server;
   forward request to Web server
   retrieve content at begin offset
   while (EOF == FALSE) {
      send content to MH
      read content from Web server;
  }
  send content to the MH
};
```

C. Implementation of the IGW

At the heart of the IGW architecture, it implements a task dispatcher component as illustrated in Fig(11), which is based on an ISAPI extension directly inherited from Microsoft *CHttpServer* class.

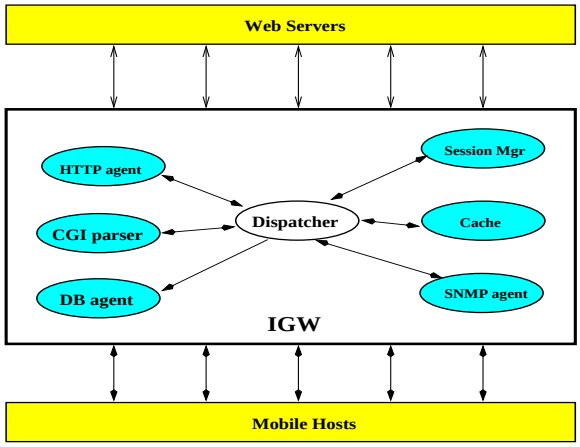

Fig. 11. Processing Flow

The task dispatcher, which is implemented on top of the Microsoft *CHttpServer* class, handles each incoming HTTP request, parses request message, and decides which action to take (which agent to invoke). An instance of *CGI* object is created upon the receipt of each HTTP request, with its pointer to an *EXTENSION_CONTROL_BLOCK* structure defined by Microsoft SDK. The CGI object implements a parser for retrieving various information embedded in the HTTP request header. The HTTP agent is then used to retrieve HTTP files from origin Web servers on behalf of MHs. In the case of HTTP failure recovery, the offset information retrieved by the CGI parser is passed to the HTTP session manager for repositioning the begin offset for file downloading. The pseudocode is given in Listing 3.

The following example is given to illustrate how this scheme works. Assume that the MH sends a request to retrieve a document with the URL http://www.cnn.com/draft.ppt. The HTTP request header is listed in Table 2:

Upon receiving the request initiated by the Web browser, the client-side proxy makes a transparent redirection to an IGW

TABLE 2
ORIGIN HTTP REQUEST HEADER

```
GET http://www.cnn.com/draft.ppt HTTP 1.0 \r\n
...
\r\n\r\n
```

with IP address 205.132.6.11, by adding session information into the HTTP request header, as showed in table 3.

TABLE 3
MODIFIED HTTP REQUEST HEADER

```
GET http://205.132.6.11/scripts/dis.dll?url=http://www.cnn.com/
draft.ppt HTTP 1.0 \r\n
User-Agent: Proxy/2.0 \r\n
Session-Offset: 0 \r\n
...
\r\n\r\n
```

Notice that *dis.dll* is the binary executable of the IGW implementation. The modified URL means that the source of content is located at *http://www.cnn.com/draft.ppt*; and that the document should be downloaded from scratch (Session-OffSet: 0). Upon receiving the request, the IGW extracts the source URL contained in the modified HTTP request header and sends the request to the origin Web server (http://www.cnn.com/draft.ppt). The response from the server will be relayed to the client-side proxy at the MH (see Fig(6) for details).

Assume that a network handoff or preemptive handoff occurs during the course of file downloading, which disrupts the ongoing HTTP session. The number of bytes received by the MH so far is assumed to be $203,223$ bytes. The byte-count information is added into an HTTP request header (offset=$203,223$), as showed in Table 4). When the client-side proxy automatically reestablishes a remote connection to the IGW, which in turn will act accordingly by fetching data from the $203,223$ bytes from the beginning of the file (draft.ppt). Data packets received by the client proxy at the MH will be properly pieced together to form a complete powerpoint document.

TABLE 4
MODIFIED HTTP REQUEST HEADER AFTER NETWORK FAILURE

```
GET http://205.132.6.11/scripts/dis.dll?url=http://www.cnn.com/
draft.ppt HTTP 1.0 \r\n
User-Agent: Proxy/2.0 \r\n
Session-Offset: 203223 \r\n
...
\r\n\r\n
```

The process of failure recovery is performed between the client proxy and the IGW in an automatic and transparent fashion without human interference.

IV. EXPERIMENTAL STUDIES

In this section, we first describe our testbed which closely resembles a real heterogeneous wireless environment, then present the experimental results of the prototype system.

The entire implementation of client-side proxy and IGW subsystems consists of roughly $20,000$ and $8,000$ lines of Visual C++ source code, respectively. On the MH, approximately $10,000$ lines of code are GUI-related, The remaining lines of code are written specifically for thread pool, HTTP parser, network awareness, and HTTP session management. On the IGW, the code primarily deals with HTTP agent, CGI parser, database agent, and HTTP session management.

The testbed consisted of two Toshiba Tecra laptops (MHs) running Microsoft XP operating system and one Dell OptiPlex desktop (IGW) running Microsoft 2000 server with 192 MB main memory and *x*86 family 6 Model CPU. Each Toshiba Tecra laptop has 1 GB main memory with an Intel Pentium III 1.2 GHz Mobile CPU and the built-in WLAN card with maximum rate of 11 Mbps. They also have a switch below the keyboard on the front, which can be used to manually turn on or off the built-in WLAN card. In addition, we installed Verizon CDMA *1xRTT* driver with the Sierra wireless AirCard 550 with maximum rate of 144 Kpbs. The routing cost metrics for the Ethernet, WLAN and CDMA interfaces are configured as 1, 2, and 5, respectively, so that the network interface with lower cost metric will be selected in the presence of multiple network interfaces.

A WavePoint II access point outfitted with 10 Dbi Omni antenna was used as an IEEE 802.11b base station to serve as a bridge between the WLAN and the wired network. It was attached to the side of a window facing east in a window office (in Applied research building in Morristown, NJ), offering a limited wireless outdoor coverage through the window. Under this testbed configuration, the IGW was accessible either from the Verizon CDMA network over 17 hops or from the WLAN over 2 hops. The entire driveway was under the coverage of the Verizon CDMA network, but only part of the driveway was under the coverage of the WLAN (see Fig(12) for details).

To evaluate the prototype system in a vehicular network environment, we conducted the experimental study focusing on the following three aspects: (1) network environment awareness, (2) transparent packet-layer HTTP failure recovery in the presence of handoffs, and (3) preemptive handoff by dynamically selecting the best network interface in the overlapping coverage area of the CDMA and the WLAN. Notice that the MHs in our testbed were always in the coverage of the Verizon CDMA *1xRTT* network, whereas the WLAN coverage relied purely on the location with respect to the WLAN's access point.

To establish a baseline comparison, we placed two MHs with/without software installation in the same vehicle. The vehicle's initial position was under the coverage area of the WLAN. We initiated a long-lived HTTP session on both MHs at the same time (downloading a 6M powerpoint file) and then drove the vehicle circling the building with speed approximately $3 \sim 7$ mph, thereby creating an intermittent WLAN connectivity during file downloading. The side-by-side comparison demonstrates that the MH having this prototype system can resynchronize with the IGW in the presence of handoffs and can dynamically select the best available network interface in the overlapping coverage area, while at the same time maintaining an user-perceived HTTP session continuity.

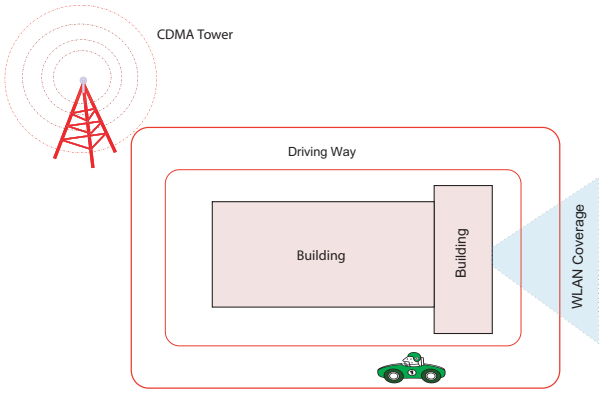

Fig. 12. Testbed Environment

In contrast, a MH without this prototype system was unable to maintain HTTP session when handoffs occurred.

A. Performance Comparison with and without preemptive handoff

It is not trivial to quantify the performance gain of the preemptive handoff in an in-vehicle environment, because the vehicle's motion, in effect, is not repeatable. To simulate the intermittent WLAN connectivity in a repeatable fashion, we conducted the experimental study in a window office with a good reception of the Verizon CDMA network.

A series of experiments under different network settings was designed to investigate the impact of network hysteresis on system throughput. First, we tested a system without network-aware capability, Secondly, we tested the prototype system using the CDMA/WLAN and CDMA/Ethernet heterogeneous networks, respectively.

In the experimental study, one MH initially had only the Verizon CDMA network connectivity. A long-lived HTTP session was initiated to download a $500K$ powerpoint document over the Verizon CDMA *1xRTT* network. After 30 seconds, a high-bandwidth network was activated by either manually plugging the Ethernet cable into the MH or turning on the MH's WLAN card, in an effort to emulate that the MH had suddenly moved into an overlapping coverage area of the WLAN. Our approach to control timing of network availability is similar to the experimental studies presented in [4], [26]. We measured and compared the elapsed time needed to retrieve the file with and without the preemptive handoff.

Fig(13) presents a snapshot of network utilization of CDMA and WLAN in the entire period of an HTTP session. It can be seen from Fig(13) that the MH initially had only CDMA network connectivity with the network utilization of 6%. Then a WLAN network was enabled at 30 seconds after an HTTP session was initiated over the CDMA network. Without network-aware and preemptive capability, the MH continued to stay with the CDMA network until the file had finished downloading despite the presence of WLAN network during the HTTP session. As a result, the network utilization of the WLAN was close to zero (Fig(13) for details). In this case, it took up 521 seconds on average to download a $500K$ powerpoint file over the CDMA network. This is because the inherent network hysteresis prevents the MH from taking advantage of the available WLAN network during the session, once the HTTP session was initially established over the CDMA network.

TABLE 5
PERFORMANCE COMPARISON IN HETEROGENEOUS NETWORK ENVIRONMENTS

Overall Time					
without preemptive		*preemptive CDMA-WLAN*		*preemptive CDMA-Ethernet*	
mean	stdev	mean	stdev	mean	stdev
$521.4s$	$59.13s$	$41.3s$	$6.34s$	$43.0s$	$2.94s$
discovery delay					
without preemptive		*preemptive CDMA-WLAN*		*preemptive CDMA-Ethernet*	
mean	stdev	mean	stdev	mean	stdev
-	-	$2.2s$	$0.24s$	$6.3s$	$2.6s$

Notice: we used a fixed IP for WLAN and a dynamically-assigned IP for Ethernet

TABLE 6
DIMINISHING BENEFIT OF PREEMPTIVE HANDOFF

WLAN		*Ethernet*		*preemptive WLAN-Ethernet*	
mean	stdev	mean	stdev	mean	stdev
41.3	$6.34s$	$33.7s$	$1.13s$	44.8	2.45

On the other hand, the MH enhanced with the preemptive handoff capability was able to sense as well as to dynamically select a higher bandwidth network during the course of HTTP sessions, resulting in a substantial saving in elapsed time of HTTP sessions. Fig(14) and Fig(15) show a spike in both WLAN, and Ethernet utilization rate after the preemptive handoff had taken place. Table 5 presents the average and standard deviation of the overall down time and discovery delay over 10 independent runs, with and without preemptive handoffs. There are two interesting points to note:

1) When both WLAN and Ethernet were configured to activate at 30 seconds after an HTTP session was initiated over the CDMA network, a preemptive handoff really took place after few seconds. Such a delay could be attributed to the amount of time required by OS to generate the connect event and to acquire an IP address dynamically assigned by a DHCP server;
2) there existed an overlapping time interval in which the MH continued to receive data packets from the CDMA network interface after it has actually closed a connection over the CDMA network and reestablished a connection over the WLAN. This portion of data was being labeled as useless traffic in Fig(15) and Fig(14). Such a phenomenon, being associated with the implementation of *TCP Time-Wait*, can be clearly explained in TCP protocol documentations *RFC 793* and *RFC 1337*.

The measurement results in Table 5 showed that in general the preemptive handoff significantly improves the system throughput in an overlapping coverage area where two networks with high bandwidth differential are involved. However, the performance advantage of the preemptive handoff might

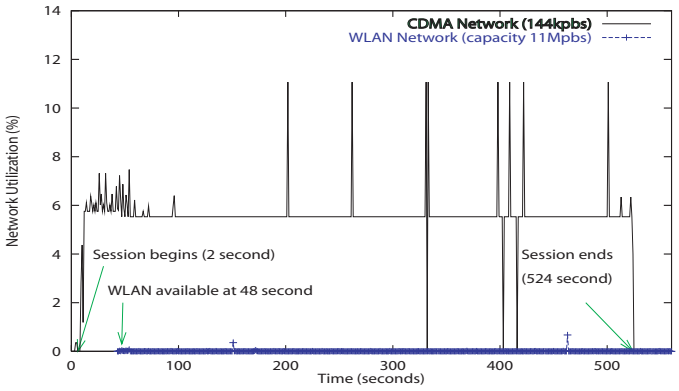

Fig. 13. Network Hysteresis (ignoring the presence of a higher bandwidth network during an HTTP session)

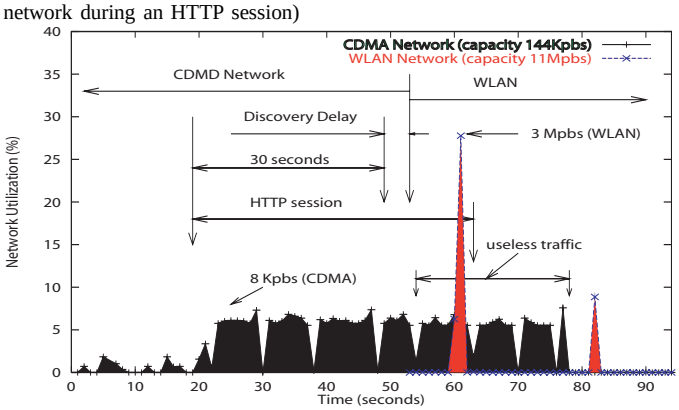

Fig. 14. Preemptive Handoff between CDMA and WLAN

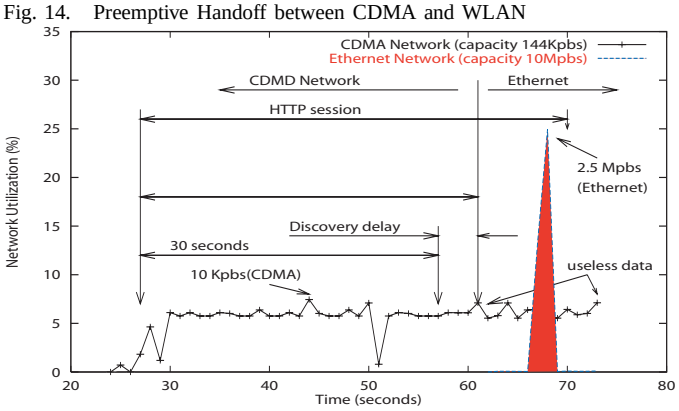

Fig. 15. Preemptive Handoff between CDMA and Ethernet

be diminished or even become negative when the two involved networks have comparable bandwidth. For instance, we performed an experiment with WLAN/Ethernet combination. As a baseline comparison, we first measured the elapsed time to download a 23M powerpoint file purely from the WLAN (11 Mbps) and the Ethernet (10 Mbps), respectively. To study the impact of preemptive handoff, one MH was initially set to have the WLAN connectivity. An HTTP session to download a 23M powerpoint file was started with the WLAN for 15 seconds, then a preemptive handoff from the WLAN to the Ethernet was triggered by manually plugging the Ethernet cable into the MH. Under this setting, the negative effect of the preemptive handoff was illustrated in Fig(16) and Table 6. This effect was observed because that it normally took a few seconds of time

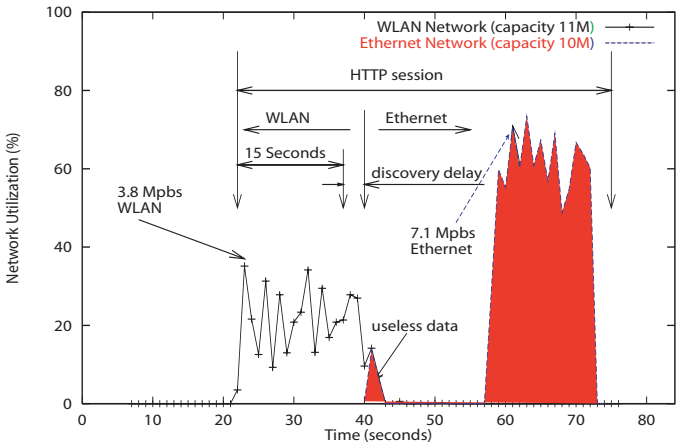

Fig. 16. Preemptive Handoff between WLAN and Ethernet

delay to complete a preemptive handoff. In general, the net benefit of the preemptive handoff can be formulated as

$$T_{est}^{A} > T_{est}^{B} + T_{handoff}, \quad (2)$$

where the subscript est denotes estimate remaining download time and the subscript $handoff$ denotes handoff time and the superscripts A and B refer to networks A and B. Eq(2) means that a preemptive handoff from network A to network B makes sense if the estimated remaining download time using network A should be greater than the estimated remaining download time using network B plus handoff time. In general, knowledge of the remaining size of an ongoing HTTP session, together with the elapsed time for network switch, is an important factor that affects the efficiency of preemptive handoff. The dynamic load of network has an effect on the end-to-end delay and actual network throughput. Hence, the real-time loads of involved networks are also an important factor for preemptive handoff, which adds another dimension of complexity.

B. Experiment for System Resilience

We conducted another experiment to test the resilience of the prototype system to random transient network disconnects. To this end, in the experimental study we artificially create a random network failure in the midst of HTTP sessions to see how the prototype system responds to it. Three cases are considered in our experiment:

1) we measured the time for downloading a 6M file via the WLAN without network failure.
2) a random network failure was produced by manually turning the AP's power off, then turning the AP's power on,
3) a random network failure was created by manually turning off the MH's WLAN card, followed by turning on the MH's WLAN card.

The first case served as a baseline for comparison study. We measured the download time of a 6M file via either a dynamical IP address assigned by the DHCP server or a static IP address.

The experimental study showed that the prototype system can survive long network disconnects triggered by manually

TABLE 7

IMPACT OF TRANSIENT FAILURES

fixed IP address					
overall download time					
No disruption		*AP power off-on*		*WLAN off-on*	
mean	stdev	mean	stdev	mean	stdev
12.3	3.16	101.9s	3.93s	43.5s	3.69s
network disconnect time					
mean	stdev	mean	stdev	mean	stdev
–	–	86.0s	4.21s	25.3s	4.4s

dynamically assigned IP address					
overall download time					
No disruption		*AP power off-on*		*WLAN off-on*	
mean	stdev	mean	stdev	mean	stdev
12.3	3.16	101.7s	1.77s	97.1s	23.5s
network disconnect time					
mean	stdev	mean	stdev	mean	stdev
–	–	84.7s	3.86s	74.7s	17.7s

turning off the MH's WLAN card or the AP's power. Table 7 showed the mean and standard deviation of the elapsed time of HTTP session over 10 independent runs. It can be seen in Table 7 and Fig(17) that even though network disconnect time lasted up to 86 seconds, the prototype system was able to automatically resynchronize with the IGW once network connectivity was resuscitated while maintaining an user-perceived HTTP session continuity.

It can be seen from Fig(17) that the MH took more than 10 seconds to capture the event that the WLAN card had been shut down manually, and only took roughly two seconds to capture the disconnect event that the WLAN card has been manually turned on. This observation is in line with Microsoft design guidance [8] which states that the NIC need to wait 10 seconds before generating any disconnect event and the connect event is generated at most 2 seconds. As an application-layer approach, the prototype system cannot control the OS kernel to improve the system responsiveness to such events. It, however, demonstrates a resilience to network disconnects that last more than 10 seconds [11], [10], [19].

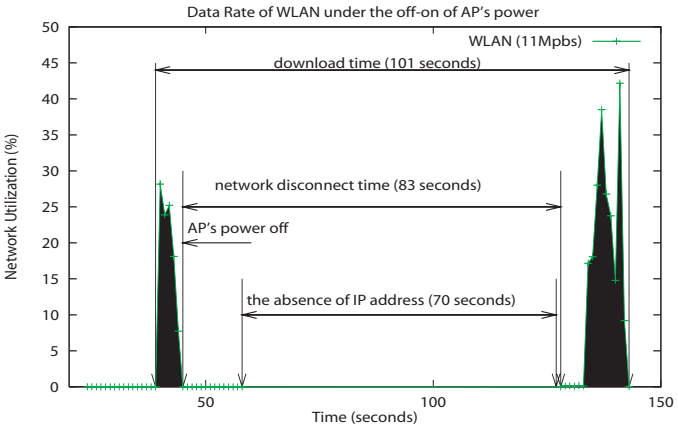

Fig. 17. HTTP Session under the off-on of AP's power

C. Evaluation of Horizontal Handoff

The goal of this experiment is to study and measure the horizontal handoff delay between two WLAN subnets as well as detection delay of the prototype system, and to evaluate the ability of the prototype system to handle such handoffs.

To measure the impact of handoffs between two WLAN subnets, we used two access points (WP-II E made by Lucent Technologies) in our testbed and each AP was connected to an IP subnet, with the beacon frequency of APs being configured as 1 second. A MH running Window XP was configured to have access to two IP subnets via a DHCP server. The MH was located in the overlapping coverage area of both APs, with one AP being configured as the preferred one and another as the backup.

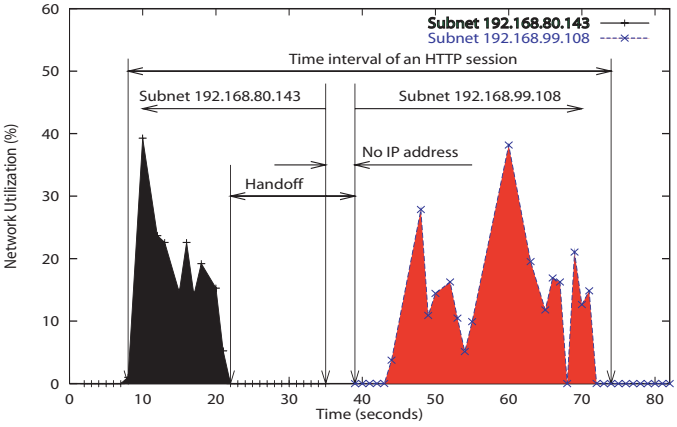

Fig. 18. Handoff Between WLAN subnets

In this experiment, we initiated a long-lived HTTP session (downloading a 23M powerpoint file) via the preferred AP, and then triggered a handoff by turning off the AP's power. This forced the MH to change its point of attachment from the preferred AP to the backup AP.

It stands to reason that data packets traveling through the preferred AP should immediately cut off when its power was off, marking a starting point for a network handoff (see Fig(18)). Such an observation would lead to a decomposition of delay into two parts: network delay and detection delay.

TABLE 8

PERFORMANCE COMPARISON

Overall download time			
No Handoff		*One Handoff*	
mean	stdev	mean	stdev
41.3s	6.34s	67.0s	3.25s
Network handoff delay			
No Handoff		*One Handoff*	
mean	stdev	mean	stdev
–	–	17.4s	3.4s
Detection delay			
No Handoff		*One Handoff*	
mean	stdev	mean	stdev
–	–	2.2s	0.42s

A horizontal handoff delay is defined as the time interval that starts when the rate of data packet associated with the

preferred AP (one IP subnet) drops to zero and ends when the MH has attached to the backup IP subnet. A detection delay, which is closely related to our implementation, is defined as the time interval that starts when the MH has attached to backup subnet and ends when the MH starts receiving and transmitting data packet via the new IP subnet. By examining the trace of IP packets, we were able to accurately measure both the horizontal handoff delay and the detection delay.

Table 8 summarized the performance measurements of the prototype system over 10 independent runs. These results indicated that a handoff across different subnets could cause a 65% additional delay in downloading a 23M file. It can be seen from Table 8 and in Fig(18) that horizontal handoff delay, which includes the amount of elapsed time to release the IP address of the preferred subnet and to acquire the IP address of the backup subnet, could take up to 17.4 seconds on average, in which the absence of IP address can last for more than 5 seconds. The average detection delay was about 2 seconds over 10 independent runs, with the standard deviation of $0.42s$ (see Table 8 for details). These experimental results highlight the importance of implementing a resilient system in order to survive long handoffs in a practical setting.

V. Conclusion and Future Work

The real challenge in vehicular environments is how to best utilize geographically dispersed WLAN networks and omnipresent cellular networks such as CDMA, particularly in overlapping coverage areas, and how to provide a transparent and efficient failure recovery mechanism with the ability to adapt dynamically changing, inherently heterogeneous network environments.

In this paper, we identify three problems that are critical to vehicular Internet access: (1) efficient HTTP failure recovery in network volatility while at the same time maintaining an user-perceived session continuity, (2) network environment awareness, and (3) network adaptation via preemptive handoff in the presence of multiple networks during HTTP sessions. We have designed and implemented a multi-tier prototype system with specifically tailored and carrier-agnostic features for vehicular Internet access.

The main objective of this paper is to address these emerging problems stemming from inherently heterogeneous and dynamically changing vehicular environments, from the user's perspective, rather than from the wireless operator's perspective. Therefore our focus is not placed on how to reduce handoff latency, but is rather placed on network awareness, network adaptation, and transparent HTTP failure recovery. This work differs significantly from prior research aiming at reducing handoff latency at layer 2 and layer 3. It is worth noting that the prototype system is an application-layer approach that can fully exploit new advances at network and physical layers.

The performance study showed that the prototype system can provide transparent and efficient HTTP failure recovery in the presence of vertical/horizontal/preemptive handoffs and is robust across transient network failures. We showed that the phenomenon of the network hysteresis prevents MHs from taking advantage of available network resources in an overlapping coverage area of WLAN and a cellular network (CDMA) and presented an application-layer preemptive handoff to mitigate the impact of the network hysteresis on the overall system throughput.

The idea behind the proposed preemptive handoff is to initiate a network handoff based on network capacity, rather than on channel characteristics or signal strength level. The advantage of preemptive handoff lies in its ability to dynamically select the best possible connection with a higher-bandwidth network in overlapping coverage areas, which could result in a substantial reduction in elapsed time for long-lived HTTP sessions in a heterogeneous network consisting of WLAN and CDMA/CDPD/GPRS.

Many research issues remain for further exploration. Decreasing handoff latency is a pressing issue that needs to be adequately addressed. Our future work will focus on how to reduce layer-two and layer-three handoff latency.

VI. Appendix

Proof: Let $T > 0$ be a polling interval. Let $\{X_i, i \geq 1\}$ be the interarrival times of a Poisson process, representing the time instants at which changes in network environment occur. It is obvious that the random variables $X_i, i \geq 1$ are independently and exponentially distributed with the mean $\frac{1}{\lambda}$ [24].

Define $S_0 = 0,\ S_n = \sum_{i=1}^{n} X_i$. It follows that S_i represents the arrival time of the *i*th change event. Let

$$n(T) = \sup\{n \geq 0 :\ S_n \leq T\}, \tag{3}$$

where $n(T)$ is the Poisson counting process with rate λ, representing the number of the event changes in the interval $(0, T]$. Define detection delay, denoted by $D(.)$, as follows:

Definition 4: Detection delay at the time t is

$$\mathrm{D}(t) = \begin{cases} 0, & \text{if polling occurs at } t \\ t - S_{n(t)}, & \text{otherwise} \end{cases}$$

It can be seen from Fig(19) that three event changes occur in the time interval T, the detection delay is thus determined by the time instant at which the last event change occurs and T, that is, $D(T) = T - S_{n(T)} = T - S_3$[1]. For any given interval $(0, t]$, the long-run mean average detection delay over the interval $(0, t]$ could be expressed as

$$\begin{aligned} E(D(\cdot)) &= \lim_{t\to\infty} \frac{E(\int_0^t D(\tau)d\tau)}{t} \\ &= \lim_{t\to\infty} \frac{E(\lfloor \frac{t}{T} \rfloor \int_0^T D(\tau)d\tau)}{t} \qquad (4) \\ &= \frac{E(\int_0^T (D(\tau))d\tau)}{T} = \frac{\int_0^T E(D(\tau))d\tau}{T}, \qquad (5) \end{aligned}$$

[1] $D_i = T - S_i$ refers to detection delay with respect to the *i*th event change in the time interval T

where Eq(4) follows the renewal reward theory [24], and Eq(5) assumes that integral and expectation operations are exchangeable.

Consider that the nth event occurs at time $s \in (0, T]$, $S_n = \sum_{i=1}^{n} X_i$. It is obvious that S_n follows the gamma distribution [24] as follows:

$$f_n(s) = \frac{\lambda^n}{(n-1)!} s^{n-1} e^{-\lambda s}, \ s > 0. \tag{6}$$

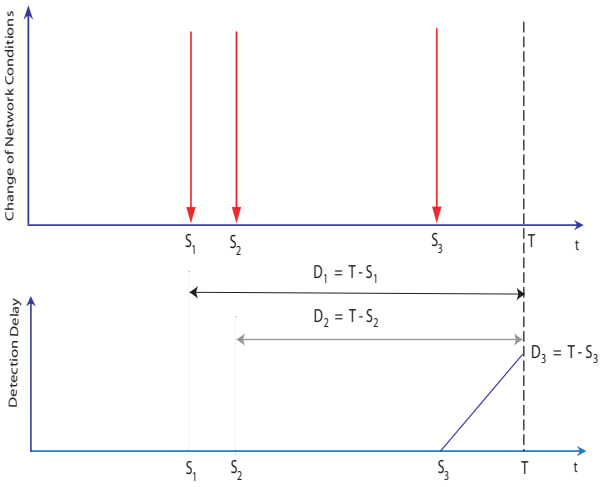

Fig. 19. Evolution of Detection Delay Function

Thus the expected detection delay at time $\tau \in (0, T]$ is written as

$$\begin{aligned} E(D(\tau)) &= \int_0^{\tau} (\tau - s) f_n(s) ds \\ &= \tau \left(1 - \sum_{i=0}^{n-1} \frac{(\lambda\tau)^i \exp(-\lambda\tau)}{i!} \right) \\ &\quad - \frac{n}{\lambda} \left(1 - \sum_{i=0}^{n} \frac{(\lambda\tau)^i \exp(-\lambda\tau)}{i!} \right). \end{aligned} \tag{7}$$

The analysis of general case could be hard since $n(\tau)$ is a random variable representing the number of change events in the interval $(0, \tau)$. Let's only consider the first event in $(0, T]$, which serves as a upper bound for general case. By inserting $n = 1$ into Eq(7) we obtain

$$E(D_1(\tau)) = \tau \left(1 - \frac{1 - \exp(-\lambda\tau)}{\lambda\tau} \right). \tag{8}$$

Substituting Eq(8) into Eq(9) yields

$$\begin{aligned} E(D_1) &= \frac{\int_0^T \tau (1 - \frac{1-\exp(-\lambda\tau)}{\lambda\tau}) d\tau}{T} \\ &= \left(\frac{(T\lambda)^2 - 2T\lambda + 2 - 2\exp(-T\lambda)}{2T\lambda^2} \right), \end{aligned} \tag{9}$$

where $E(D_1)$ denotes the detection delay by only considering the first event occurring in the interval $[0, T]$.

Based on Eq(3), we have that $D_n = T - S_n \leq D_1 = T - S_1$ for $1 \leq n$, implying that $E(D_1) \leq E(D_n)$. The proof thus is completed. ▲

VII. Acknowledgment

The authors would like to thank the anonymous reviewers for their critically reviewing the manuscript and for their truly helpful constructive comments. Yibei Ling would like to thank Dr. Shu-Chan Hsu in the Department of Cell Biology and Neuroscience at Rutgers University for her encouragement and support. The preliminary material in this paper was presented in part at the 2004 IEEE International Conference on Networking Sensing and Control.

References

[1] Wireless Application Protocol White Paper. In *http://www.wapforum.org/what/WAP_white_pages.pdf*. WAP Forum, June 20 2000.

[2] Introducing Tarantella. In *http://www.tarantella.com/support/documentation/enterprise/eus/base/gettingstarted/tarantella_intro.html*. Sun Microsystem, 2005.

[3] Farooq Anjum and Ravi Jain. Performance of TCP over Lossy Upstream and Downstream Links with Link Level Retransmission. In *Proceedings of the USENIX 1993 Winter Conference*, January 1999.

[4] M. Baker, X. Zhao, S. Cheshire, and J. Stone. Supporting mobility in mosquitonet. In *Proc. of the 1996 USENIX Conference*, January 1996.

[5] Ramon Caceres and Liviu Ifode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5):850–857, June 1995.

[6] S. Cheshire and M. Baker. Experiences with a wireless network in mosquitonet. In *Proc. of the 1995 IEEE Hot Interconnects Symposium*, August 1995.

[7] K. Claffy, G. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. In *Proceedings of INER's 1998 Conference*, July 1998.

[8] Microsoft Corporation. IEEE 802.11 Network Adapter Design Guidances for Windows XP. *Microsoft Corporation*, December 2002.

[9] Minoru Katayama et.al. A method of achieving service continuity between different networks. *IEICE Transactions on Communications (in Japanese)*, J84-B(3):452–460, 2001.

[10] N.A. Fikouras, K. El Malki, S. R. Cvetkovic, and C. Smythe. Performance Evaluation of TCP Over Mobile IP. In *Proc. PIMRC*, 1999.

[11] Anne Fladenmuller and Ranil De Silva. The Effect of Mobile IP Handoff on the Performance of TCP. *Mobile Networks and Application*, 4(2):131–135, May 1999.

[12] Stathes Hadjiefthymiades, Stamatis Papayiannis, and Lazaros Merakos. Using path prediction to improve tcp performance in wireless/mobile environments. *IEEE Coomunications Magazine*, 40(8):54–61, August 2002.

[13] Georgios Karagiannis and Geert Heijenk. Mobility Support for Ubiquitous Internet Access. In *ERICSSON Open Report*. http://www.ub.utwente.nl/webdocs/ctit/1/00000038.pdf, 2000.

[14] Bil Lewis and Daniel J. Berg. *Threads Primer: A Guide to Multithreaded Programming*. SunSoft Press, New York, 1996.

[15] Yi-Bing Lin, Per-Chun Lee, and I. Chlamtac. Dynamic Periodic Location Area Update in Mobile Networks. *IEEE Trans. on Veh. Technol*, 51(6):1494–1501, 2002.

[16] Yibei Ling, Wai Chen, Russell Hsing, and Onur Altinta. Network Awareness and Adaptation. In *IEEE International Conference on Networking, Sensing and Control*, March 2004.

[17] Yibei Ling, Tracy Mullen, and Xiaola Lin. Analysis of Optimal Thread Pool Size. *ACM Operating System Review*, 34(2):42–55, 2000.

[18] M.Ylianttila, M. Pande, J. Mkel, and P. Mhnen. Optimization Scheme for Mobile Users Performing Vertical Handoffs between IEEE 802.11 and GPRS/EDGE Networks. In *Proceedings of IEEE Global Telecommunications Conference*, volume 6, Auguest 2001.

[19] Fikouras N, El Malki K, Cvetkovic SR, and Smythe C. Performance of TCP and UDP during Mobile IP Handoffs in Single-agent Subnetworks. In *Proc. IEEE Wireless Communications and Networking Conference*, 1999.

[20] Tadasho Okoshi, Masahiro Mochizuki, Yoshito, and Hideyuki Tokuda. Mobilesocket: Toward Continuous Operation for Java Application. In *IEEE 8th International Conference on Computer Communications and Networks*, 1999.

[21] Christina Parsa and J.J. Garcia-Luna-Aceves. Improving TCP Performance over Wireless Networks at the Link Layer. In *Proceedings of 7th International Conference on Network Protocols*, 1999.

[22] Charles E. Perkins. *Mobile Networking Through Mobile IP*. Sun Microsystems, New York, 1998.

[23] Charles E. Perkins and Kuang-Yeh Wang. Optimized Smooth Handoffs in Mobile IP. In *Proceedings of the The Fourth IEEE Symposium on Computers and Communications*, 1999.

[24] Sheldon M. Ross. *Stochastic Processes*. John Wiley & Sons, Inc., New York, 1996.

[25] Henning Schulzrinne. SIP for Mobility Application. http://www.cs.columbia.edu/ hgs/sip/talks/von0006_schulzrinne2.pdf, June 20 2000.

[26] Mark Stemm and Randy H. Katz. Vertical Handoffs in Wireless Overlay Networks. *Mobile Networks and Applications*, 3(4):335–350, 1998.

[27] Elin Wedlun and Henning Schulzrinne. Mobility Support using SIP. In *The Second ACM/IEEE International Conference on Wireless and Mobile Multimedia*, August 1999.